In the claims:

Claim 1. (currently amended) A modeling method for defining software applications using a visualizable computer executable modeling language, said method comprising:

providing a plurality of display elements for displaying screen objects on a display screen;

displaying components corresponding to said screen objects;

defining each of the software applications as a hierarchy of process models, slots, data models and flow rules;

classifying some of said process models and said data models as atomic;

classifying all other process models and data models as composite;

defining each of said composite process models as a construction of at least one of sub process models, slots, data models and flow rules;

defining a portion of said slots as having one of the following sub-classifications:

mandatory; and

optional;

defining each of said composite data models as a construction of at least one sub data model; and

defining each of said flow rules as connecting a pair of said slots, data models and sub data models, wherein said flow rules define both data flow and process flow, such that the need for writing code in any programming language to implement the software applications is eliminated.

Claim 2. (currently amended) A visualizable computer executable modeling language system operating in accordance with the method of claim 1, for a ~~substantially~~ complete definition of the software applications, said system comprising:

process models, each of which may contain any number of sub process models, slots, data models and flow rules;

data models, each of which may contain any number of sub data models; and

flow rules, each of which connecting a pair of said slots, data models and sub data models, thereby defining data flow and process flow,

wherein said process models and data models and slots and flow rules are arranged in a structural hierarchy conforming to a set of rigid composition rules, ensuring the language system ~~is rich enough and precise enough for~~ enables a computer to execute an application model defined in said modeling language.

Claim 3. (original) The modeling language system of claim 2, further comprising at least one visual representation.

Claim 4. (original) The modeling language system of claim 3, wherein said visual representation comprises:

process diagrams comprising various two dimensional shapes representing said process models;

sub process diagrams comprising various two dimensional shapes contained within said process diagrams, representing said sub process models;

slot diagrams comprising various two dimensional shapes situated on the edges of said process diagrams and said sub process diagrams, representing said slots;

data trees comprising hierarchical tree structures contained within said process diagrams, representing said data models and said sub data models; and

flow arrows comprising arrows connecting pairs of said slot diagrams, said data trees and sub-trees of said data tress, said arrows representing said flow rules.

Claim 5. (original) The modeling language system of claim 2,

wherein each of said slots is further defined as having one of the following classifications: input slot; and output slot (exit); and further defining each of said input slots as having one of the following sub-classifications: synchronous input slot (trigger); and asynchronous input slot; and further defining each of said triggers as having one of the following sub-classifications: mandatory; and optional; and further defining some of said exits as having the sub-classification terminating.

Claim 6. (original) The modeling language system of claim 2, wherein each of said slots is further defined as having one of the following classifications:

input slot; and

output slot (exit);

and further defining each of said flow rules contained in said composite process model as connecting one source and one target; and further defining the source of each of said flow rules to be one of the following:

an input slot of said composite process model; an exit of a sub process model of said composite process model;

a data model of said composite process model; and

a sub data model of a data model of said composite process model;

and further defining the target of each of said flow rules to be one of the following:

an exit of said composite process model;

an input slot of a sub process model of said composite process model;

a data model of said composite process model; and

a sub data model of a data model of said composite process model.

Claim 7. (original) The modeling language system of claim 2, wherein:

each of said process models may further contain a reference to a database table (process table);

at least some of the sub data models of data models of said process model are marked as interesting fields; and

each of said interesting fields further contains a reference to a column of said process table.

Claim 8. (original) The modeling language system of claim 7, wherein:

a selection condition of an SQL query (addressing clause) may be attached to an input slot of said process model to select matching instances of said process model each time data is to be received by said instances through said input slot;

said addressing clause is defined in terms of a matching condition between the interesting fields of said process model and the data model of said data to be received through said input slot.

Claim 9. (original) The modeling language system of claim 2, wherein each of said process models may further contain a reference to a computer code implementing the function of said process model.

Claim 10. (original) The modeling language system of claim 2, wherein:

each of said composite data models is composed of said sub data models by one of the following structure means: concatenation; collection; and selection, each of said sub data models having a classification as one of the following: mandatory; and optional; each of said sub data models may further be marked as recurring, with a further optional indication of minimal and maximal number of occurrences;

each of said data models may further contain constraints on the data it defines, comprising at least one of the following:

legal characters; and minimal and maximal length; each of said data models may further comprise a set of legal values and an initial value; and

each of said data models may further comprise formatting directives.

Claim **11**. (original) A modeling system for defining software applications using the visualizable computer executable modeling language system of claim **2**, enabling users to create, display, modify and test, in an integrated workspace, models of said modeling language, in accordance with the rules of said modeling language system, wherein: said modeling system comprises a graphical user interface tool (visual modeling tool) for creating, displaying, modifying and testing models of said modeling language in an integrated workspace, such that users of said modeling tool create and edit said models using various graphical user interface (GUI) operations.

Claim **12**. (original) The modeling system of claim **11**, wherein each of said process models and said data models is further defined as having one of the following classifications: dependent model: only exists as a sub model of a specific parent model; and reusable model: may be reused as a sub model of multiple parent models, wherein each of said reusable models is assigned a unique identifier.

Claim **13**. (original) The modeling system of claim **11**, wherein models are formally represented as one of the following:

Extensible Markup Language (XML) documents;

structured database records; and any other equivalent binary representation, and wherein a repository of said representations of said models, arranged as a hierarchy of packages and sub-packages (knowledge base), is used to maintain libraries of said models, and wherein the modeling system

displays said models whose said representations are stored in said knowledge base, stores in said knowledge base said representations of new said models that are defined by the users of the modeling system, and updates said representations of said models in said knowledge base according to modifications made to said models by said users.

Claim **14.** (original) The modeling system of claim **11**, further comprising at least the following editing capabilities: selection of editing operations from menus; adding components to said models through dragging of models from palettes of existing models; and modifying attributes of said models and components of said models.

Claim **15.** (currently amended) The modeling system of claim **11**, wherein said workspace comprises a ~~virtually infinite~~ drawing board for displaying hierarchies of two dimensional visual diagrams, each representing a corresponding said hierarchy of models, and wherein said users are able to zoom in and out from a currently displayed part of said hierarchy of diagrams, enabling the display of the details of said model and any sub-model thereof at any desired level of said hierarchy of models.

Claim **16.** (original) The modeling system of claim **11**, further comprising: a software program (runtime engine) to execute models defined in said modeling language; and a visual debugger for testing and debugging said models, wherein: said runtime engine, as it executes said models, produces records listing the details of said execution (trace events); said trace events are used to record and store the history of said execution; and said visual debugger uses said stored trace events to display the current status of instances of processes, including the content

of their data, as well as the processing steps that have led to said current status.

Claim 17. (currently amended) A software program (runtime engine) to execute applications defined in the visualizable computer executable modeling language system of claim 2, wherein:

each of the applications is defined by a single said process model and a hierarchy of its sub-models; and

the runtime engine executes the application exactly as defined by said single process model and said hierarchy of its sub-models, thus ~~substantially~~ eliminating the need for writing code in any programming language to implement the application.

Claim 18. (original) The runtime engine of claim 17, further comprising: active models comprising objects responsible for representing and enacting the definitions and rules embodied in said models, where there is an active model corresponding to each of said process models and data models; runtime objects comprising objects containing the runtime state of instances of said process models and data models, where there may be at any time any number of runtime objects instantiated from each of said process models and data models by the corresponding said active model; and a model loader comprising an object responsible for loading said models from their formal representations stored in a repository, converting said loaded models to corresponding said active models, and caching said active models.

Claim 19. (original) The runtime engine of claim 17, wherein the runtime engine executes each of said models as a series of processing steps, wherein: each of said processing steps is triggered by the receipt of an external input; the runtime engine invokes at least one instance of at least one relevant

process model to handle said received input, and executes sub-processes of said invoked processes as defined by the relevant said flow rules; and a processing step ends when any further activities to be performed depend on the receipt of other external inputs.

Claim **20**. (original) The runtime engine of claim **17**, wherein: the runtime engine executes each of said models as a series of processing steps by invoking at least one instance of said process models; the full state of each of said at least one process instance is made persistent at the end of each said-processing step; execution of each of said at least one process instance can resume from its stored state at any relevant time; and a repository of all said at least one process instances is available for queries and retrieval by the runtime engine while executing said models or by external applications.

Claim **21**. (currently amended) A software program (code generator) to generate the code of a software program implementing the applications defined in the visualizable computer executable modeling language system of claim **2**, wherein:

each of the applications is defined by a single said process model and a hierarchy of its sub-models; and

the code generator produces code in a general purpose programming language implementing the application exactly as defined by said single process model and said hierarchy of its sub-models, thus ~~substantially~~ eliminating the need for writing code in any programming language to implement the application.

Claim **22**. (original) The software program of claim **21**, wherein said general purpose programming language is Java.

Claim **23**. (original) The software program of claim **21**, wherein said general purpose programming language is C++.

Claim **24**. (currently amended) A method for ~~substantially~~ overcoming the need to write computer source code in order to develop software applications, comprising:

creating models of the applications in a visualizable computer executable modeling language system, using a visual modeling tool, comprising:

defining each of said software applications as a hierarchy of process models, slots, data models and flow rules;

classifying some of said process models and said data models as atomic;

classifying all other process models and data models as composite;

defining each of said composite process models as a construction of at least one of sub process models, slots, data models and flow rules;

defining each of said composite data models as a construction of at least one sub data model; and

defining each of said flow rules as connecting a pair of said slots, data models and sub data models, wherein said flow rules define both data flow and process flow; and

executing the logic defined by said created models.

Claim **25**. (original) The method of claim **24**, wherein the execution of the logic defined by said models is made by a dedicated computer program (runtime engine).

Claim **26**. (original) The method of claim **24**, wherein the implementation of the logic defined by said models is made by the code of a software program in a general purpose programming

language, which is generated by a dedicated computer program (code generator).

Claim 27. (currently amended) A software development platform for ~~substantially~~ overcoming the need to write computer source code in order to develop software applications, comprising:

a visualizable computer executable modeling language for the definition of software solutions, said definition comprising:

defining each of said software solutions as a hierarchy of process models, slots, data models and flow rules;

classifying some of said process models and said data models as atomic;

classifying all other process models and data models as composite;

defining each of said composite process models as a construction of at least one of sub process models, slots, data models and flow rules;

defining each of said composite data models as a construction of at least one sub data model; and

defining each of said flow rules as connecting a pair of said slots, data models and sub data models, wherein said flow rules define both data flow and process flow;

a visual modeling tool for defining said software solutions by at least one user as said hierarchies of models in said modeling language; and

a dedicated computer program to automatically execute said software solutions according to the logic defined by said hierarchies of models.

Claim 28. (original) The software development platform of claim 27, wherein said dedicated computer program is a runtime engine that automatically executes said software solutions at runtime, according to the logic defined by said hierarchies of models.

Claim 29. (original) The software development platform of claim 27, wherein said dedicated computer program is a code generator that automatically generates the code of a software program in a general purpose programming language implementing the logic defined by said hierarchies of models.

Claim 30. (new) A computer program product comprising a computer usable medium having computer readable code embodied therein for execution on a general purpose computer, said computer usable medium storing instructions that, when executed by the computer, cause the computer to perform a modeling method for defining software applications using a visualizable computer executable modeling language, said method comprising:

providing a plurality of display elements for displaying screen objects on a display screen;

displaying components corresponding to said screen objects;

defining each of the software applications as a hierarchy of process models, slots, data models and flow rules;

classifying some of said process models and said data models as atomic;

classifying all other process models and data models as composite;

defining each of said composite process models as a construction of at least one of sub process models, slots, data models and flow rules;

defining a portion of said slots as having one of the following sub-classifications:

mandatory; and

optional;

defining each of said composite data models as a construction of at least one sub data model; and

defining each of said flow rules as connecting a pair of said slots, data models and sub data models, wherein said flow rules define both data flow and process flow,

such that the need for writing code in any programming language to implement the software applications is eliminated.